合肥工業大學

操作系统实验报告

实验题目 实验 8 分页存储器管理

学生姓名 袁焕发

学　　号 2019217769

专业班级 物联网工程 19-2 班

指导教师 田卫东

完成日期 2021 年 11 月 24 日

1. 实验目的和任务要求

学习 i386 处理器的二级页表硬件机制，理解分页存储器管理原理。

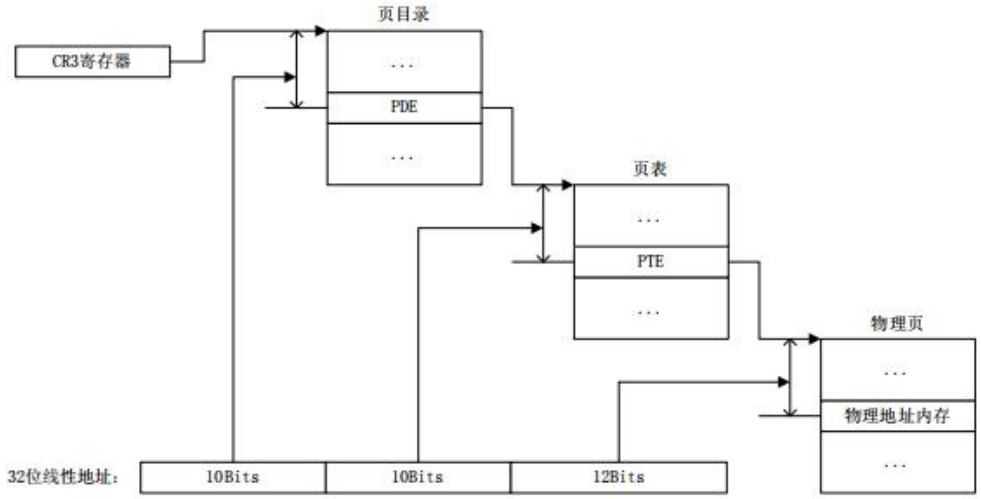
查看 EOS 应用程序进程和系统进程的二级页表映射信息，理解页目录和页表的管理方式。

编程修改页目录和页表的映射关系，理解分页地址变换原理。

2. 实验原理

EOS 中的每个进程都有一个独立的 4G 虚拟地址空间（即 4GB 的逻辑地址空间），其中低 2G 为进程私有的用户地址空间，高 2G 为所有进程共享的系统地址空间。进程的用户地址空间用于存放用户进程的代码、数据等。系统地址空间被 EOS 内核使用，用于存放内核的代码、内核运行时的各种数据结构以及所有线程的内核模式栈等。

在二级页表分页机制中，无论是页目录(Page Directory)、页表(Page Table)还是物理页，它们的大小均为 4KB。其中，第一级是一个页目录，在这个唯一的页目录中包含了 1024 个页目录项(PDE，Page Directory Entry)。页目录中的每个页目录项可以映射第二级的一个页表，所以，页目录最多可以映射 1024 个页表。第二级的每个页表都包含了 1024 个页表项(PTE，Page Table Entry)，每个页表项可以映射一个 4KB 大小的物理页。这样，一个页目录可以映射 1024 个页表，每个页表又可以映射 1024 个物理页，所以，二级页表最终就可以映射 1024*1024*4K=4GB 的物理地址。



3. 实验内容

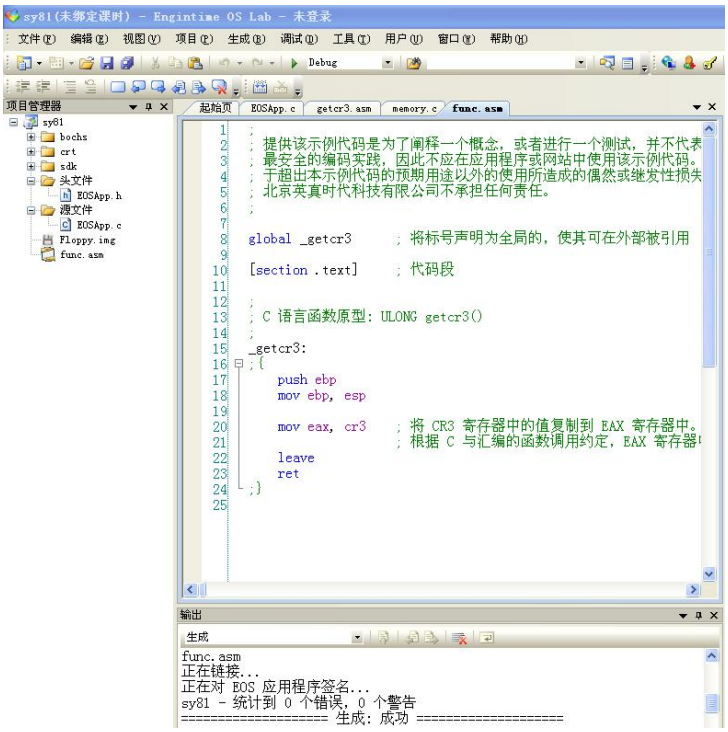
3.1. 准备实验

新建一个 EOS Kernel 项目，使用 Debug 配置和 Release 配置生成此项目的 EOS SDK。新建一个 EOS 应用程序项目，使用在第 3 步生成的 SDK 文件夹覆盖 EOS 应用程序项目文件夹中的 SDK 文件夹。

3.2. 查看 EOS 应用程序进程的页目录和页表

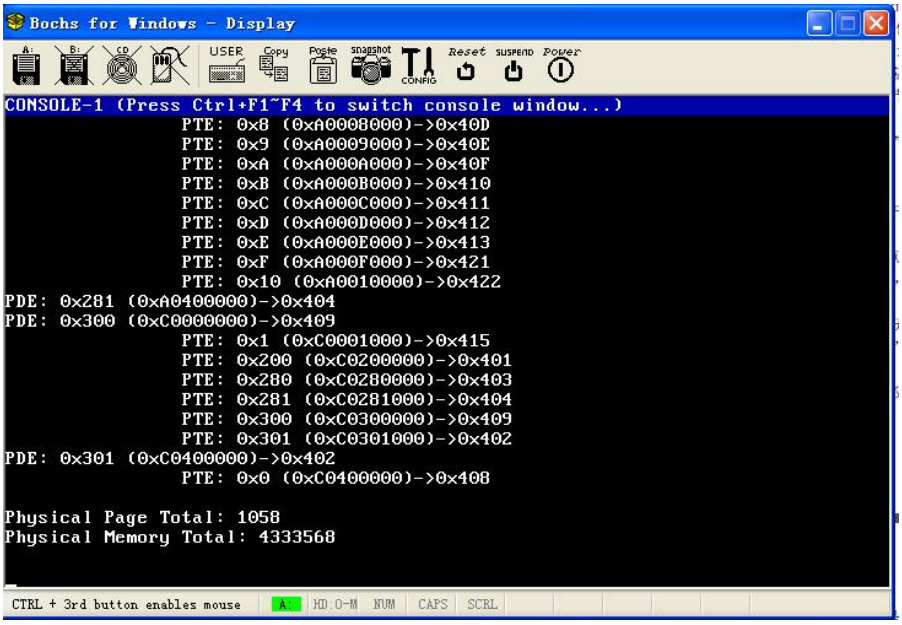
打开“学生包”中本实验对应的文件夹，使用 OS Lab 打开其中的 memory.c 和 getcr3.asm 文件。右键点击“项目管理器”窗口中的“源文件”，在弹出的

快捷菜单中选择“添加”中的“添加新文件”，选择“asm 源文件”模板，输入文件名称“func”，将 getcr3.asm 文件中的源代码复制到 func.asm 文件中，按 F7 生成修改后的 EOS 应用程序项目。

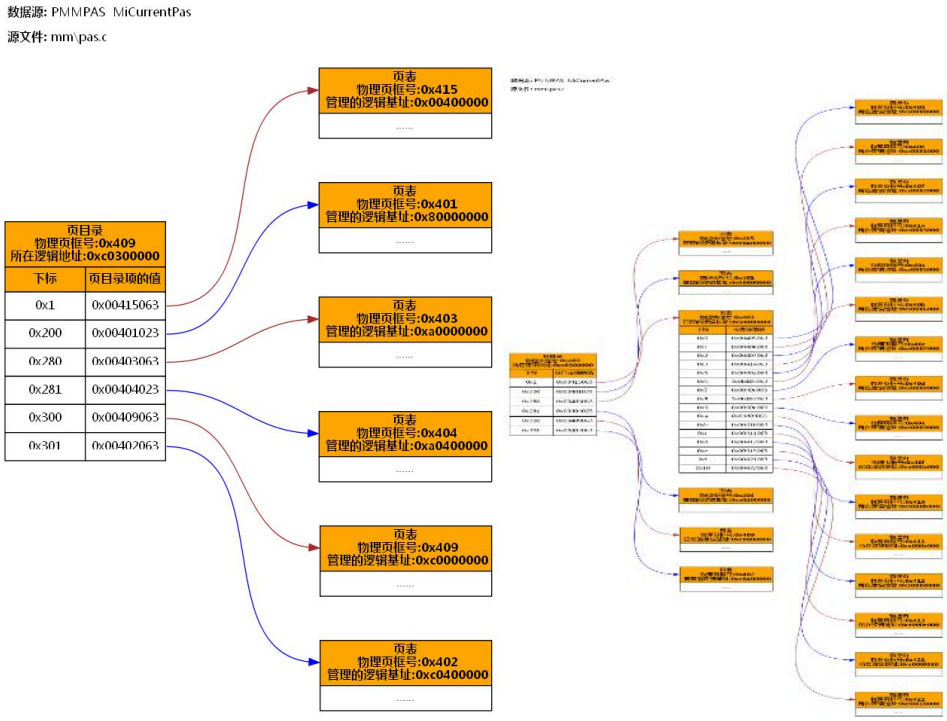


在 main 函数的返回代码处（第 190 行）添加一个断点，按 F5 启动调试。

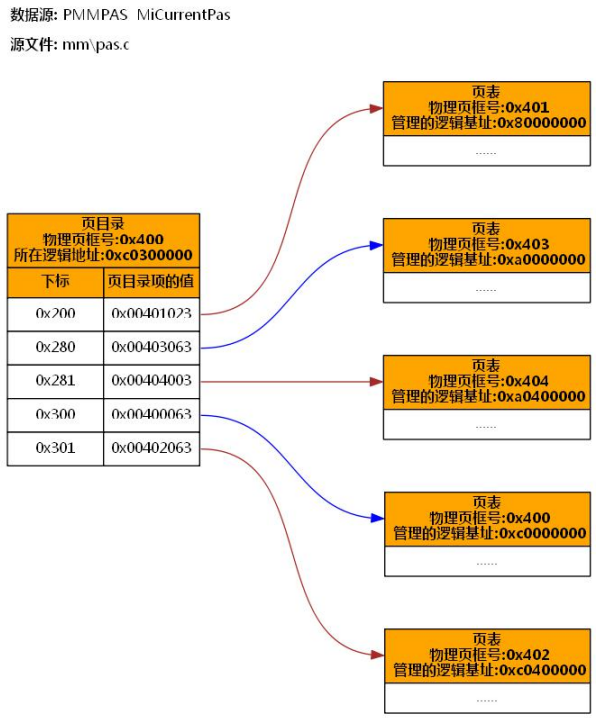
EOS 启动完毕后会自动运行应用程序，将应用程序进程的页目录和页表打印输出到屏幕上，然后在刚刚添加的断点处中断。



选择“调试”菜单“窗口”中的“二级页表”，打开“二级页表”窗口，点击该窗口工具栏上的“刷新”按钮，可以查看页目录和页表的映射关系。



“二级页表”窗口默认只显示页目录和页表的信息，为了查看页表映射的物理页，可以点击“二级页表”窗口工具栏上的“绘制页表映射的物理页”按钮，在打开的对话框中，可以选择页表中指定的页表项映射的物理页，然后点击“绘制”按钮，就可以绘制出页表项到物理页的映射了。



应用程序进程的页目录和页表一共占用了几个物理页？页框号分别是多少？

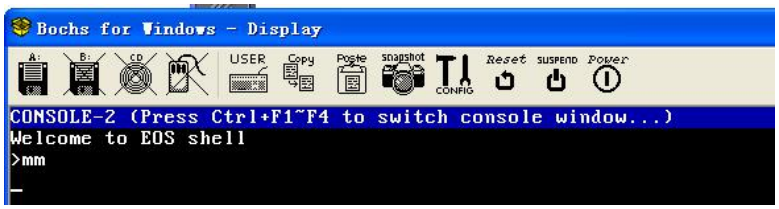
答：页目录占用一个物理页，页框号是 0x409；页表占用 5 个物理页，页框号分别是是 0x41D、0x401、0x403、0x404、0x402。

映射用户地址空间（低 2G）的页表的物理页框号是多少？该页表有几个有效的 PTE，或者说有几个物理页用来装载应用程序的代码和数据？物理页框号分别是多少？

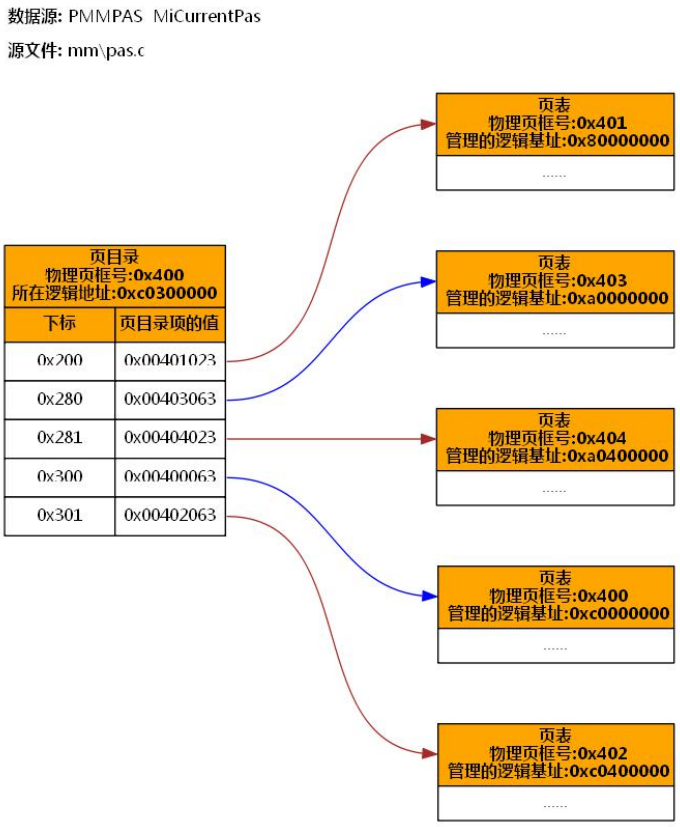
答：映射用户地址空间的页表的页框号是 0x41D；该页表有 11 个有效的 PTE；物理页框号分别是 0x41E、0x41F、0x420、0x421、0x422、0x423、0x424、0x425、0x426、0x427、0x428。

3.3. 查看应用程序进程和系统进程并发时系统进程的页目录和页表

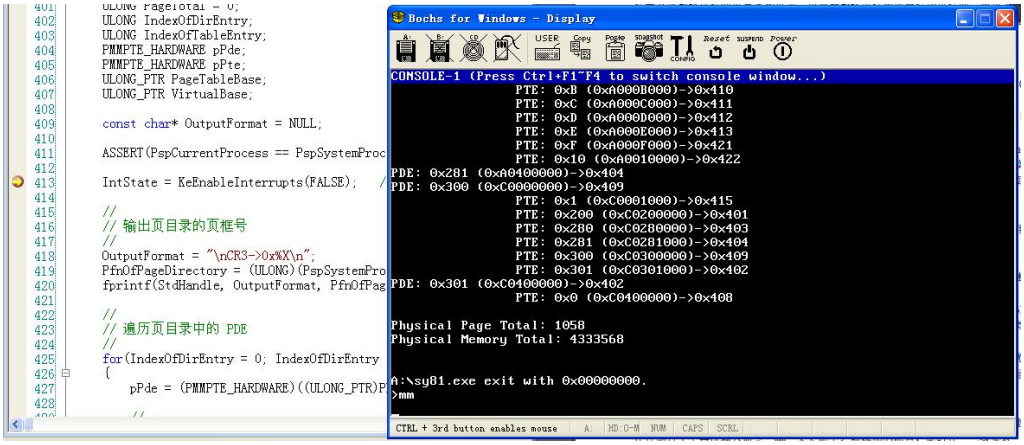
结束之前的调试，并删除之前添加的所有断点，取消 EOSApp.c 文件第 113 行语句的注释，生成修改后的 EOS 应用程序项目。将 sysproc.c 文件拖动到 OS Lab 窗口中释放，在 ConsoleCmdMemoryMap 函数中的第 413 行代码处添加一个断点。按 F5 启动调试 EOS 应用程序，在“Console-1”中会自动执 EOSApp.exe，创建该应用程序进程。利用其等待 10 秒的时间，按 Ctrl+F2 切换“Console-2”。输入命令“mm”后按回车，程序在断点处中断。



刷新“二级页表”窗口，窗口中显示的是系统进程的二级页表映射关系。



按 F5 继续调试，待 mm 命令执行完毕后，切换到控制台 1，此时 应用程序又会继续运行，待其结束后，在控制台 1 中再次输入命令“mm”又会命中之前添加的断点。



EOS 启动后系统进程是一直运行的，所以当创建应用程序进程后，系统中就同时存在了两个进程，这两个进程是否有各自的页目录？在页目录映射的页表中，哪些是应用程序进程和系统进程共享的，哪些是独占的？分析其中的原因。
答：系统进程和应用程序进程一定有各自的页目录；映射了用户地址空间的页表被应用程序进程独占，页框号是 0x41D，映射了内核地址空间的页表都是共享的。

统计当应用程序进程和系统进程并发时，总共有多少物理页被占用？
答：应用程序进程占用 1066 物理页，系统进程页目录占用 1 物理页，总共 1067 物理页被占用。

思考为什么系统进程（即内核地址空间）占用的物理页会减少？
答：应用程序结束后，EOS 内核会删除应用程序进程在内核地址空间中占用的内存，例如删除 PCB 对象等。这些内存必须要回收，否则如果一个应用程序反复运行多次，内核空间就有可能被耗尽，操作系统就失去了可靠性。

3.4. 查看应用程序进程并发时的页目录和页表

结束之前的调试，并删除之前添加的所有断点。取消 EOSApp.c 文件第 188 行语句的注释，生成修改后的 EOS 应用程序项目，在 EOSApp.c 文件的 main 函数中的第 115 行代码处添加一个断点。启动调试，待 EOS 启动完成后，会在控制台 1 中自动运行 EOS 应用程序创建应用程序的第一个进程，并在开始的位置等待 10 秒钟，迅速按 Ctrl+F2 切换到控制台 2，并在控制台 2 中输“eosapp”后按回车（请根据应用程序可执行文件的名称调整命令），再使用该应用程序创建第二个进程，此时，应用程序创建的两个进程就开始并发运行了。刷新“进程线程”窗口，可以看到当前除了系统进程之外，还有两个应用程序进程。其中，第一个应用程序进程的主线程处于运行状态，第二个应用程序进程的主线程处于阻塞状态。

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
2	24	N	8	1	26	"A:/sy81.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Running (2)	24	0x8001f97e PspProcessStartup

PspCurrentThread

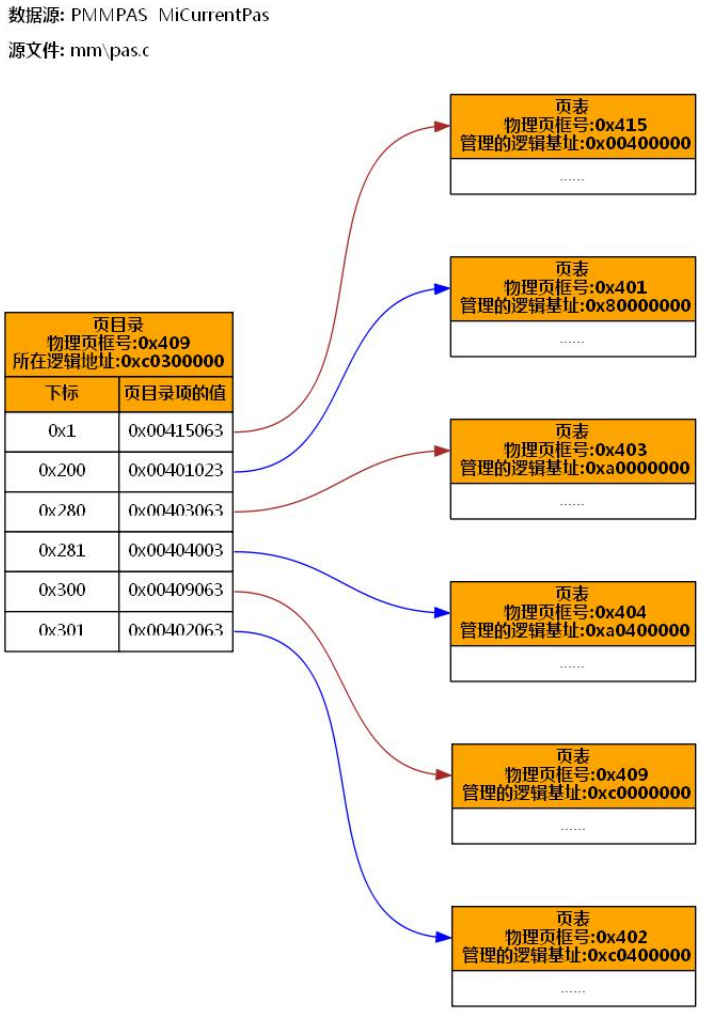
进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
3	27	N	8	1	29	"a:/sy81.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	29	N	8	Waiting (3)	27	0x8001f97e PspProcessStartup

由于当前正在运行的是应用程序的第一个进程，所以在刷新“二级页表”窗口后，可以查看第一个应用程序进程的二级页表映射。



按 F5 继续调试，等待一段时间后，又会命中刚刚添加的断点。刷新“进程线程”窗口，这次是第一个应用程序进程的主线程处于阻塞状态，说明其打印输出完毕后，在 main 函数中第 188 行处调用 Sleep 函数发生了阻塞；而第二个应用程序进程的主线程处于运行状态，也就是说当前中断运行的是第二个应用程序进程的主线程。

数据源: POBJECT_TYPE PspProcessType、POBJECT_TYPE PspThreadType

源文件: ps\psobject.c

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
1	1	Y	24	6	2	"N/A"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	2	Y	0	Ready (1)	1	0x80017e40 KiSystemProcessRoutine
2	17	Y	24	Waiting (3)	1	0x80015724 IopConsoleDispatchThread
3	18	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
4	19	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
5	20	Y	24	Waiting (3)	1	0x80017f4b KiShellThread
6	21	Y	24	Waiting (3)	1	0x80017f4b KiShellThread

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
2	24	N	8	1	26	"A:/sy81.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	26	N	8	Waiting (3)	24	0x8001f97e PspProcessStartup

进程列表

序号	进程 ID	系统进程 (System)	优先级 (Priority)	线程数量 (ThreadCount)	主线程ID (PrimaryThreadID)	镜像名称 (ImageName)
3	27	N	8	1	29	"a:/sy81.exe"

线程列表

序号	线程 ID	系统线程 (System)	优先级 (Priority)	状态 (State)	父进程 ID (ParentProcessID)	起始地址与函数名 (StartAddress And FuncName)
1	29	N	8	Running (2)	27	0x8001f97e PspProcessStartup ← PspCurrentThread

假设进程一使用的 0x416 和 0x417 物理页保存了应用程序的可执行代码，由于可执行代码通常是不变的、只读的，现在假设优化过的 EOS 允许同一个应用程序创建的多个进程可以共享那些保存了可执行代码的物理页，尝试结合图 16-3 写出共享可执行代码的物理页后进程二用户地址空间的映射信息。并说明共享可执行代码的物理页能带来哪些好处？

答：共享可执行代码的物理页可以提高运行效率高，因为进程可以直接读写内存，而不需要任何数据的拷贝。

前面的问题只是解决了共享可执行代码物理页的问题，其实就算是易变的数据所占用的物理页也是可以共享的。例如在创建进程二时，可以使之共享进程一中那些还没有发生过写操作的数据页，然后当进程一或进程二对共享的数据页进行写操作时，必须先复制一个新的数据页映射到自己的进程空间中，然后再完成写操作，这就是“写时复制”（Copy on write）技术。请读者进一步说明使用“写时复制”技术能带来哪些好处。感兴趣的读者也可以尝试在 EOS 操作系中实现“写时复制”技术。

答：写时复制是一种计算机程序设计领域的优化策略。其核心思想是，如果有多

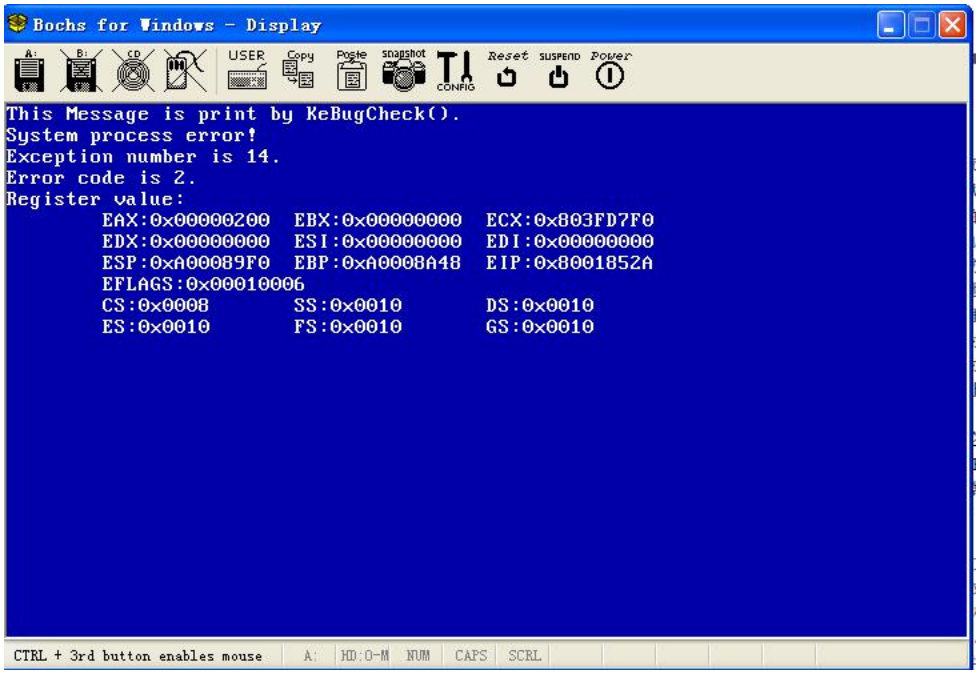
个调用者同时请求相同资源（如内存或磁盘上的数据存储），他们会共同获取相同的指针指向相同的资源，直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本给该调用者，而其他调用者所见到的最初的资源仍然保持不变。这过程对其他的调用者都是透明的。写时复制的优点是如果调用者没有修改该资源，就不会有副本被建立，因此多个调用者只是读取操作时可以共享同一份资源

统计当两个应用程序进程并发时，总共有多少物理页被占用？有更多的进程同时运行呢？根据之前的操作方式，尝试在更多的控制台中启动应用程序来验证自己的想法。如果进程的数量足够多，物理内存是否会用尽，如何解决该问题？

答：以进程 1 占用的物理页为基准，进程 2 共有 14 个物理页与其不同，1069＋14＝1083，即总共有 1083 物理页被占用。如果有更多的进程同时运行，就会有更多的物理页被占用，由于物理页的数量是有限的，物理内存会被用尽。可以使用虚拟内存技术解决该问题。

3.5. 在系统进程的二级页表中映射新申请的物理页

新建一个 EOS Kernel 项目，打开 ke/sysproc.c 文件。打开“学生包”本实验对应的文件夹中的 MapNewPage.c 文件（将文件拖动到 OS Lab 窗口中释放即可）。在 sysproc.c 文件的 ConsoleCmdMemoryMap 函数中找到“关中断”的代码行（第 413 行），将 MapNewPage.c 文件中的代码插入到“关中断”代码行的后面，生成该内核项目，启动调试。在 EOS 控制台中输入命令“mm”后按回车，EOS 会出现蓝屏。

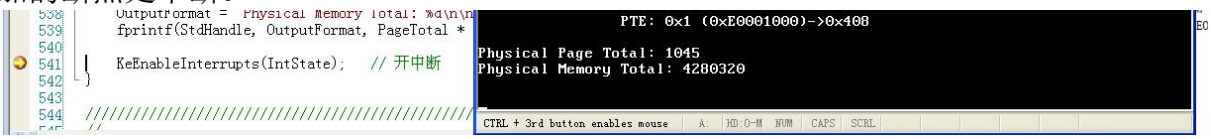


并显示错误原因是由于 14 号异常（缺页异常）引起的。原因就是由于刚刚从 MapNewPage.c 文件复制的第二行代码所访问的虚拟地址没有映射物理内存，所以对该虚拟地址的访问会触发缺页异常，而此时 EOS 还没有为缺页异常安装中断服务程序，所以就调用 KeBugCheck 函数显示蓝屏错误了。

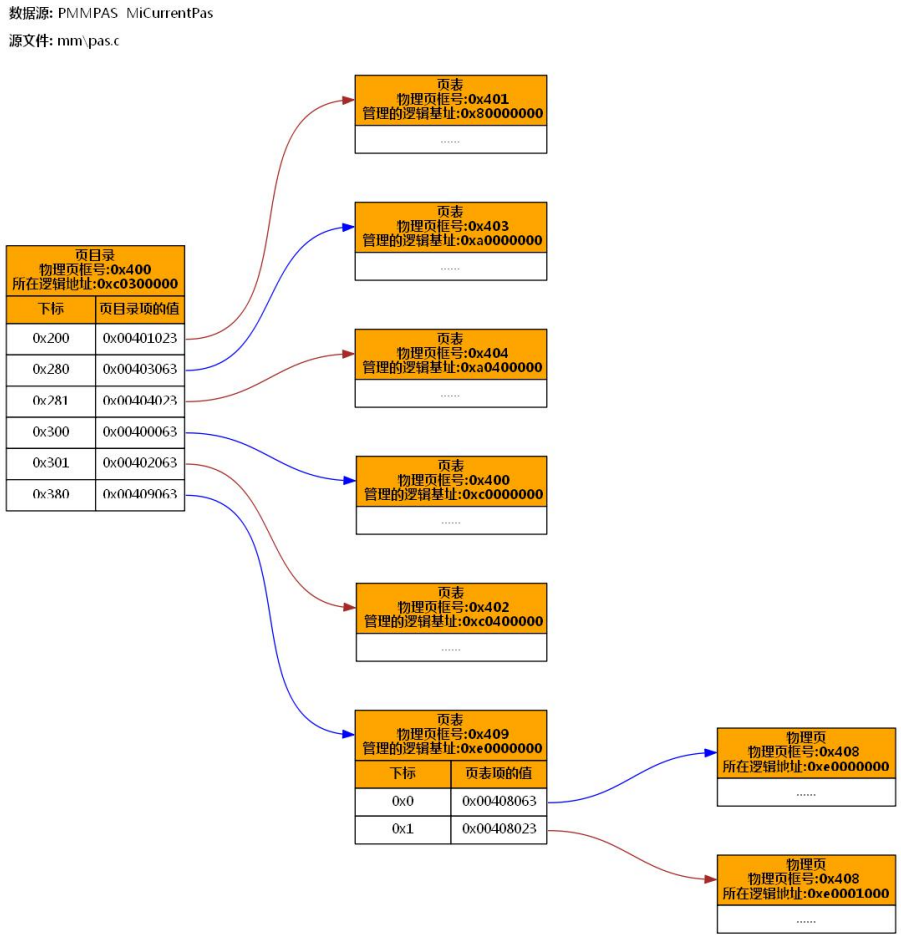
结束此次调试，然后删除或者注释掉会触发异常的那行代码。

```
418 //
419 // 访问未映射物理内存的虚拟地址会触发异常。
420 // 必须注释或者删除该行代码才能执行后面的代码。
421 //
422 //*((PINT)0xE0000000) = 100;
423
424 //
425 // 从内核申请两个未用的物理页。
426 // 由 PfnArray 数组返回两个物理页的页框号。
427 //
```

生成该内核项目，在 sysproc.c 文件的 ConsoleCmdMemoryMap 函数中打开中断的第 539 行处添加一个断点，按 F5 启动调试。在 EOS 控制台中输入命令“mm”后按回车，当在控制台窗口中输出二级页表映射信息完毕后，会在刚刚添加的断点处中断。



点击“二级页表”窗口工具栏上的“绘制页表映射的物理页”按钮，在弹出的对话框中，从下拉控件中选择“PDE 0x380 映射的页表”项目，然后勾选该页表包含的两个页表项，点击“绘制”按钮，可以查看指定页表项到物理页的映射，如图所示。



4. 实验的思考与问题分析

4.1. 观察之前输出的页目录和页表的映射关系，可以看到页目录的第 0x300 个 PDE 映射的页框号就是页目录本身，说明页目录被复用为了页表。而恰恰就是这种映射关系决定了 4K 的页目录映射在虚拟地址空间的 0xC0300000-0xC0300FFF,4M 的页表映射在 0xC0000000-0xC03FFFFF。现在，假设修改了页目录，使其第 0x100 个 PDE 映射的页框号是页目录本身，此时页目录和页表会映射在 4G 虚拟地址空间的什么位置呢？说明计算方法。

答：

PDE标号0x100作为虚拟地址的高10位，PTE标号0x100作为虚拟地址的12-22位，得到虚拟地址0x40100000。 页表：PDE标号0x100作为虚拟地址的高10位，PTE标号0x0作为虚拟地址的12-22位，得到虚拟地址0x40000000。

4.2. 修改 EOSApp.c 中的源代码，通过编程的方式统计并输出用户地址空间占用的内存数目。

答：

4.3. 修改 EOSApp.c 中的源代码，通过编程的方式统计并输出页目录和页表的数目。注意页目录被复用为页表。

答：

4.4. 在 EOS 启动时，软盘引导扇区被加载到从 0x7C00 开始的 512 个字节的物理内存中，计算一下其所在的物理页框号，然后根据物理内存与虚拟内存的映射关系得到其所在的虚拟地址。修改 EOSApp.c 中的源代码，尝试将软盘引导扇区所在虚拟地址的 512 个字节值打印出来，与 boot.lst 文件中的指令字节码进行比较，验证计算的虚拟地址是否正确。

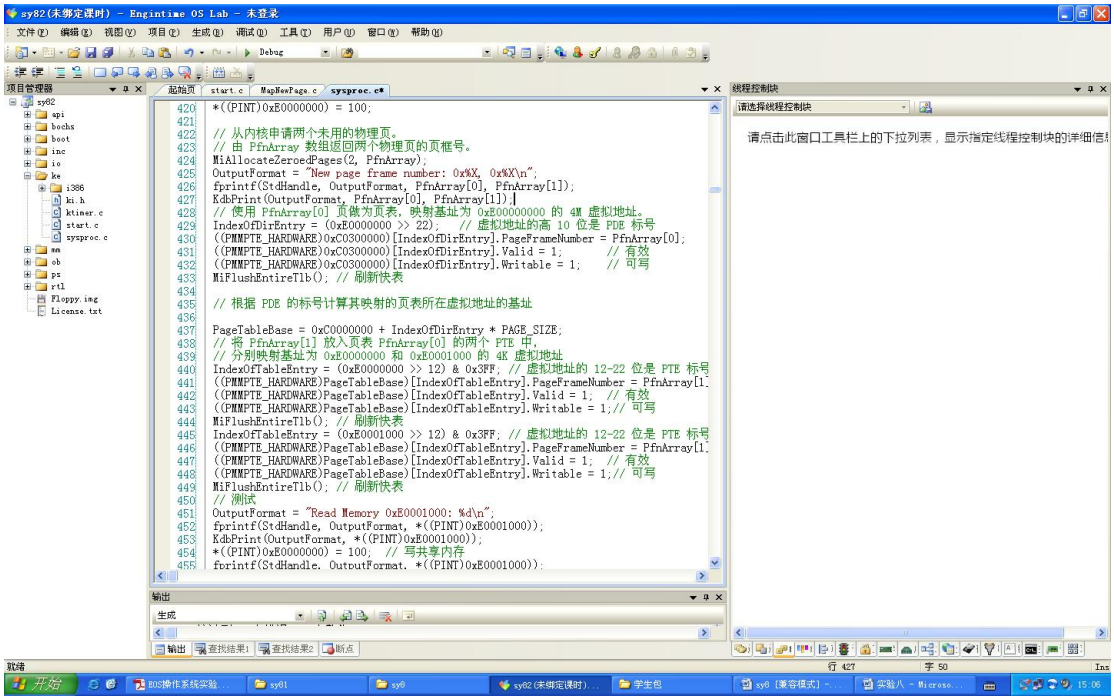
答：参考实验二中，loader.lst 的指令字节码比较过程，验证虚拟地址的值。

4.5. 既然所有 1024 个页表（共 4M）映射在虚拟地址空间的 0xC0000000-0xC03FFFFF，为什么不能从页表基址 0xC0000000 开始遍历，来查找有效的页表呢？而必须先在页目录中查找有效的页表呢？编写代码尝试一下，看看会有什么结果。

答：

不能从页表基址 0xC0000000 开始遍历查找有效的页表因为:只有当一个虚拟地址通过二级页表映射关系能够映射到实际的物理地址时，该虚拟地址才能够被访问，否则会触发异常。由于并不是所有的页表都有效，所以不能从页表基址 0xC0000000 开始遍历。

4.6. 学习 EOS 操作系统内核统一管理未用物理页的方法（可以参考本书第 6 章的第 6.5 节）。尝试在本实验第 3.5 节中 ConsoleCmdMemoryMap 函数源代码的基础上进行修改，将申请到的物理页从二级页表映射中移除，并让内核回收这些物理页。



4.7. 待完成实验 14 中的写文件功能后，可以改进本实验中的应用程序的源代码和 mm 命令的源代码，将应用程序进程或者系统进程的二级页表映射信息在打印输出到屏幕上的同时，也写入一个文本文件中。

答：未进行实验 14。

4.8. 思考页式存储管理机制的优缺点。

答：

优点：页式存储管理机制不要求作业或进程的程序段和数据在内存中连续存放，解决了碎片问题；提供了内存和外存统一管理的虚存实现方式，使用户可以利用的存储空间大大增加。

缺点：要求有相应的硬件支持，增加了机器成本；增加了系统开销，例如缺页中断处理机；请求调页算法如选择不当有可能产生抖动现象。

5. 总结和感想体会

通过本次实验，我了解了 i386 处理器的二级页表硬件机制，通过阅读资料明白了 PDE 和 PTE 中各个数据位的具体含义。在完成 EOS 操作系统的分页存储器管理实验后，对于分页存储器管理方式有了更深的了解，知道了它可以解决存储碎片问题。使用动态重定位方法，需要移动大量信息从而浪费处理机时间,代价比较高,因为这要把作业安置在一连续存储区内。如果允许一个进程直接分散的装入很多不相邻的分区，则无需紧凑，节省开销。